哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：栈结构及其应用

实验题目：算术表达式求值（算术计算器）

实验日期：2023.9.21

班级：2237102

学号：2022111654

姓名：李宸

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 李秀坤 |

**一、实验目的**

本实验主要要求从文件中正确识别，读取，并显示中缀表达式，并以栈的结构将中缀表达式转换为后缀表达式，并求后缀表达式的值。最后将后缀表达式及其结果显示并保存到文件中。

1. **实验要求及实验环境**

实验要求：

1. 从文本文件输入任意一个语法正确的（中缀）表达式，显示并保存该表达式。
2. 利用栈结构，把（中缀）表达式转换成后缀表达式，并以适当的方式展示栈的状态变化过程和所得到的后缀表达式。
3. 利用栈结构，对后缀表达式进行求值，并以适当的方式展示栈的状态变化过程和最终结果。

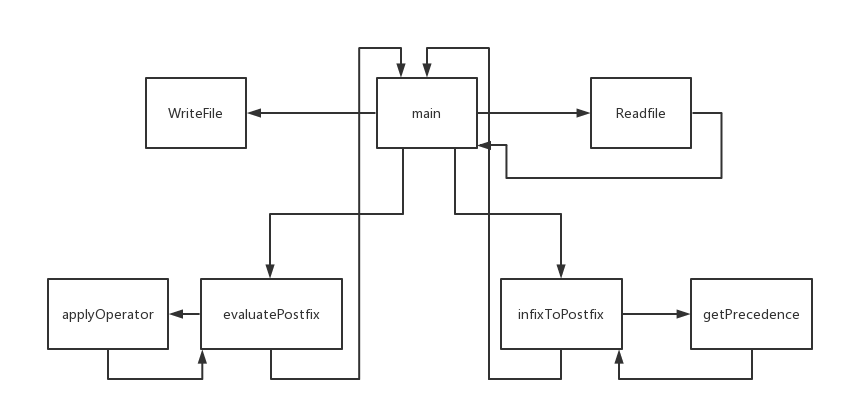
实验环境：win11 vscode1.71.2version

1. **设计思想**（本程序中的用到的所有数据类型的定义，各程序模块之间的调用关系、核心算法的流程图或主要步骤）
2. 数据类型定义及函数定义

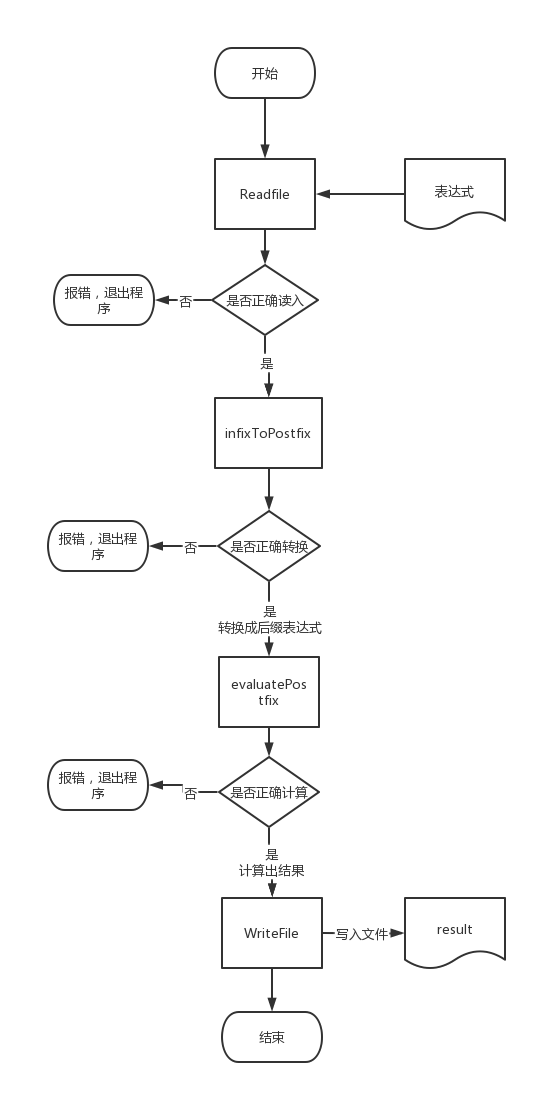
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变量 | | 数据类型 | 功能 | |
| infixExpression[] | | char | 存储中缀表达式 | |
| postfixExpression[] | | char | 存储后缀表达式 | |
| postfixIndex | | int | 指示postfixExpression的位置 | |
| numStr[] | | char | 合并，存储运算数 | |
| numStrIndex | | int | 指示numStr的位置 | |
| num | | double | 存储合并后的运算数 | |
| file | | FILE\* | 进行文件操作 | |
| length2 | | int | 记录result[]的长度 | |
| ch | | char | 从文件中逐个读取字符 | |
| start\_found | | int | 是否找到开始符 | |
| end\_found | | int | 是否找到结束符 | |
| result | | double | 记录运算结果 | |
| 函数名 | 类型 | | | 功能 |
| Readfile | void | | | 进行从文件读入的操作 |
| Writefile | void | | | 进行写入文件的操作 |
| getPrecedence | int | | | 定义运算符优先级 |
| applyOperator | double | | | 执行计算 |
| infixToPostfix | void | | | 将中缀表达式转换为后缀表达式 |
| evaluatePostfix | double | | | 计算后缀表达式的值 |
| main | int | | | 主函数 |

1. 逻辑设计

函数间调用关系：

****

核心算法流程图：

****

1. 物理设计

本程序定义了两个栈：

运算符栈：

char operatorStack[MAX\_STACK\_SIZE];

int operatorTop = -1;

其中MAX\_STACK\_SIZE为栈的最大长度；定义了char型字符串operatorStack，用于存储运算符；int型operatorTop指示栈顶元素。

运算数栈：

double valueStack[MAX\_STACK\_SIZE];

int valueTop = -1;

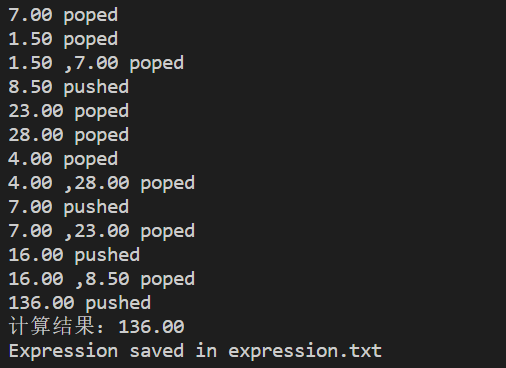
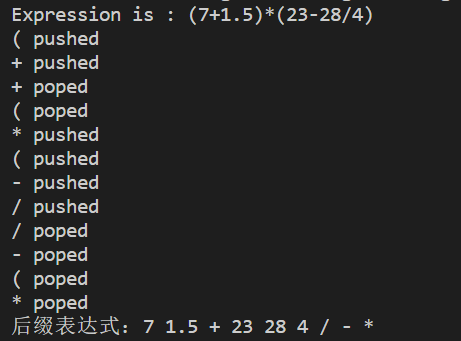
其中MAX\_STACK\_SIZE为栈的最大长度；定义了double型数组valueStack，用于存储操作数；int型valueTop指示栈顶元素。

其余变量的设计见变量表。

**四、测试结果**

1.测试数据：#(7+1.5)\*(23-28/4)#

2.测试结果：



**五、经验体会与不足**

通过后缀，中缀表达式的转换及求值，我加深了对栈结构及其应用的理解。在对字符型变量的逐个读入过程中，我学习到如何识别并处理起止符，运算数，运算符，并将运算数类型转换成浮点型。

不足：没有对首个数字为负的情况进行考虑。改进：添加if语句判断第一个字符是否是’-’，若是，则接下来第一个操作数取负。

1. **附录：源代码（带注释）**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX\_STACK\_SIZE 100

#define MAX\_EXPRESSION\_SIZE 100

// 定义操作符栈和结果栈

char operatorStack[MAX\_STACK\_SIZE];

double valueStack[MAX\_STACK\_SIZE];

int operatorTop = -1;

int valueTop = -1;

void Readfile(char expression[])//进行从文件读入的操作

{

FILE \*file;

char ch; //从文件中逐个读取字符

int start\_found = 0; //是否找到开始符

int end\_found = 0; //是否找到结束符

int i = 0;

file = fopen("expression.txt", "r");//以只读模式打开文件

if (file == NULL)

{

printf("Open file failed!\n");

return;

}

while ((ch = fgetc(file)) != EOF)//逐字符读取文件

{

if (ch == '#')

{

if (start\_found == 0)

{

start\_found = 1;

continue;//跳出本次循环，即从#后的第一个元素开始读入

}

else

{

end\_found = 1;

break; //找到结束符，结束循环

}

}

if (start\_found)//在找到起始符的情况下开始逐字符读入表达式

{

expression[i] = ch;

i++;

}

}

expression[i] = '\0';// 添加字符串结束符

fclose(file);// 关闭文件，防止内存泄露

if (start\_found && end\_found)//同时找到起止符

{

printf("Expression is : %s\n", expression);

}

else

{

printf("Incorrect # !\n");

}

}

void WriteFile(char infixExpression[], char postfixExpression[], double result)//进行写入文件的操作

{

FILE \*file = fopen("result.txt", "w");//只写模式向文件写入数据

if (file != NULL)

{

fprintf(file, "infixExpression:%s\n", infixExpression);

fprintf(file, "postfixExpression: %s\n", postfixExpression);

fprintf(file, "Expression result: %.2f\n", result);

fclose(file); // 关闭文件以防内存泄露

printf("Expression saved in expression.txt\n");

}

else

{

printf("Failed to save expression in file!\n");

}

}

// 定义运算符优先级

int getPrecedence(char op) {

if (op == '+' || op == '-') {

return 1;

} else if (op == '\*' || op == '/') {

return 2;

}

return 0;

}

// 执行计算

double applyOperator(char op, double b, double a) {

switch (op) {

case '+':

return a + b;

case '-':

return a - b;

case '\*':

return a \* b;

case '/':

if (b == 0) {

fprintf(stderr, "错误：除数为零\n");

exit(EXIT\_FAILURE);

}

return a / b;

default:

fprintf(stderr, "错误：未知的运算符 '%c'\n", op);

exit(EXIT\_FAILURE);

}

}

// 将中缀表达式转换为后缀表达式

void infixToPostfix(const char \*infix, char \*postfix) {

int postfixIndex = 0;

for (int i = 0; infix[i] != '\0'; i++) {

if (isdigit(infix[i]) || (infix[i] == '.' && isdigit(infix[i + 1]))) {

// 如果是数字或小数点，将连续的字符组合为一个数

while (isdigit(infix[i]) || infix[i] == '.') {

//printf("%c pushed\n" ,infix[i]);

postfix[postfixIndex++] = infix[i++];

}

postfix[postfixIndex++] = ' ';

i--; // 回退一个字符，以便检测运算符

} else if (infix[i] == '(') {

// 如果是左括号，直接入栈

printf("%c pushed\n" ,infix[i]);

operatorStack[++operatorTop] = infix[i];

} else if (infix[i] == ')') {

// 如果是右括号，弹出操作符栈中的操作符并加入后缀表达式，直到遇到左括号

while (operatorTop >= 0 && operatorStack[operatorTop] != '(') {

printf("%c poped\n" ,operatorStack[operatorTop]);

postfix[postfixIndex++] = operatorStack[operatorTop--];

postfix[postfixIndex++] = ' ';

}

// 弹出左括号

if (operatorTop >= 0 && operatorStack[operatorTop] == '(') {

printf("( poped\n");

operatorTop--;

} else {

fprintf(stderr, "错误：括号不匹配\n");

exit(EXIT\_FAILURE);

}

} else if (infix[i] == ' ' || infix[i] == '\t') {

// 忽略空格和制表符

continue;

} else {

// 如果是运算符，比较运算符优先级，如果栈顶运算符优先级大于等于当前运算符，出栈并输出到后缀表达式

while (operatorTop >= 0 && operatorStack[operatorTop] != '(' && getPrecedence(infix[i]) <= getPrecedence(operatorStack[operatorTop])) {

printf("%c poped\n" ,operatorStack[operatorTop]);

postfix[postfixIndex++] = operatorStack[operatorTop--];

postfix[postfixIndex++] = ' ';

}

printf("%c pushed\n" ,infix[i]);

operatorStack[++operatorTop] = infix[i];

}

}

// 将操作符栈中的剩余操作符加入后缀表达式

while (operatorTop >= 0) {

if (operatorStack[operatorTop] == '(' || operatorStack[operatorTop] == ')') {

fprintf(stderr, "错误：括号不匹配\n");

exit(EXIT\_FAILURE);

}

printf("%c poped\n" ,operatorStack[operatorTop]);

postfix[postfixIndex++] = operatorStack[operatorTop--];

postfix[postfixIndex++] = ' ';

}

postfix[postfixIndex] = '\0';

}

// 计算后缀表达式的值

double evaluatePostfix(const char \*postfix) {

for (int i = 0; postfix[i] != '\0'; i++) {

if (isdigit(postfix[i]) || (postfix[i] == '.' && isdigit(postfix[i + 1]))) {

// 如果是数字或小数点，将连续的字符组合为一个数并入栈

char numStr[MAX\_STACK\_SIZE];

int numStrIndex = 0;

while (isdigit(postfix[i]) || postfix[i] == '.') {

numStr[numStrIndex++] = postfix[i++];

}

numStr[numStrIndex] = '\0';

double num = atof(numStr);

printf("%.2f poped\n",num);

valueStack[++valueTop] = num;

i--; // 回退一个字符，以便检测运算符

} else if (postfix[i] == ' ') {

// 忽略空格

continue;

} else {

// 如果是运算符，弹出栈顶的两个数进行计算，并将结果入栈

double b = valueStack[valueTop--];

double a = valueStack[valueTop--];

printf("%.2f ,%.2f poped\n",b,a);

double result = applyOperator(postfix[i], b, a);

printf("%.2f pushed\n",result);

valueStack[++valueTop] = result;

}

}

// 最终栈中应只剩下一个结果

if (valueTop == 0) {

return valueStack[valueTop];

} else {

fprintf(stderr, "错误：操作数不匹配\n");

exit(EXIT\_FAILURE);

}

}

int main() {

char infixExpression[MAX\_EXPRESSION\_SIZE];

char postfixExpression[MAX\_EXPRESSION\_SIZE];

Readfile(infixExpression);

infixToPostfix(infixExpression, postfixExpression);

printf("后缀表达式：%s\n", postfixExpression);

double result = evaluatePostfix(postfixExpression);

printf("计算结果：%.2f\n", result);

WriteFile(infixExpression,postfixExpression,result);

return 0;

}